Элементы технического зрения ПРТС

ИНТЕГРИРОВАНИЕ ФУНКЦИЙ. ИНТЕГРИРОВАНИЕ УРАВНЕНИЙ.

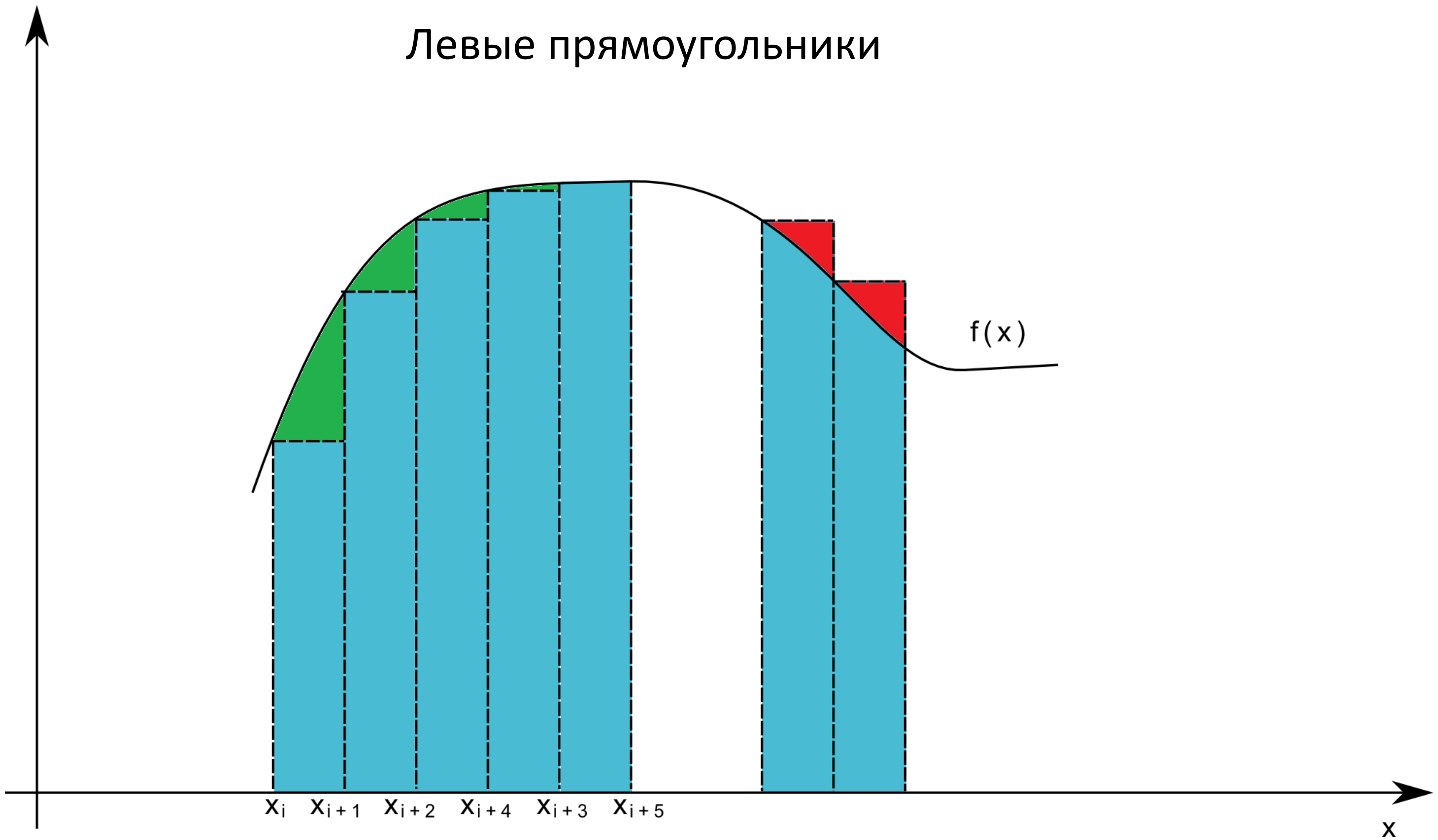
Ввод из файла

```
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>

using namespace std;

int doSomething (std::vector<double>& tempreture, std::vector<int>& displayed);

int main()
{
    fstream inputFile;
    inputFile.open("Testdata.txt");

    std::vector<double> tempreture;
    std::vector<int> displayed;

    char buffer [1024];
    double fpValue;
    int intValue;
    while (inputFile.good())
    {
        inputFile >> buffer;
        cout << buffer << "\t";
        intValue = atoi(buffer);
        inputFile >> buffer;
        cout << buffer << endl;
        fpValue = atof(buffer);
        tempreture.push_back(fpValue);
        displayed.push_back(intValue);
    }
    doSomething(tempreture, displayed);

    return 0;
}
```

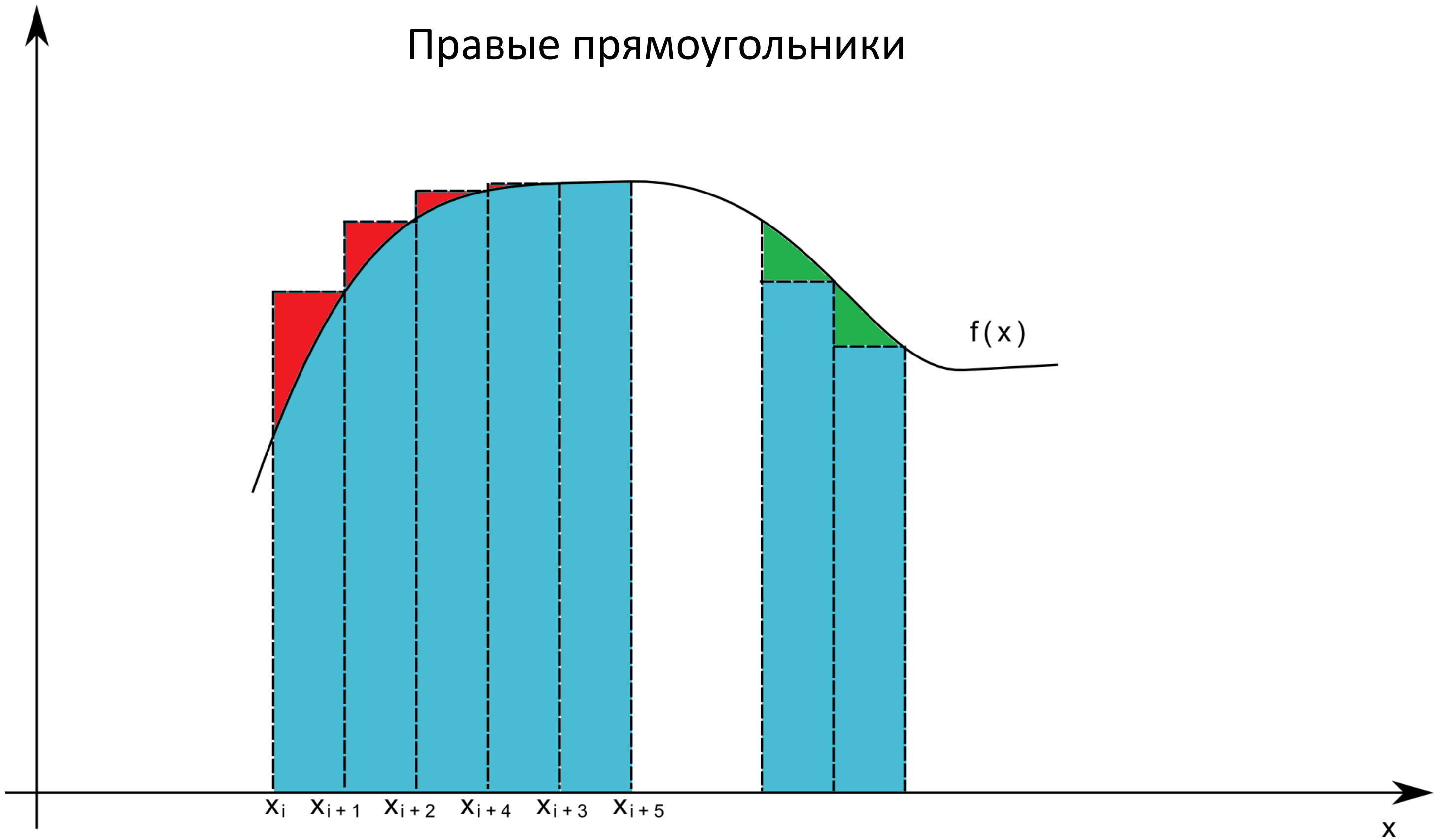
Стандартные контейнеры

```
12      37.5
18      39.3
19      42.4
20      46.3
21      51.1
3       1.1
2       -3.0
1 experiment: displayed=12 temp=37.5
2 experiment: displayed=18 temp=39.3
3 experiment: displayed=19 temp=42.4
4 experiment: displayed=20 temp=46.3
5 experiment: displayed=21 temp=51.1
6 experiment: displayed=3 temp=1.1
7 experiment: displayed=2 temp=-3
Press <RETURN> to close this window...
```

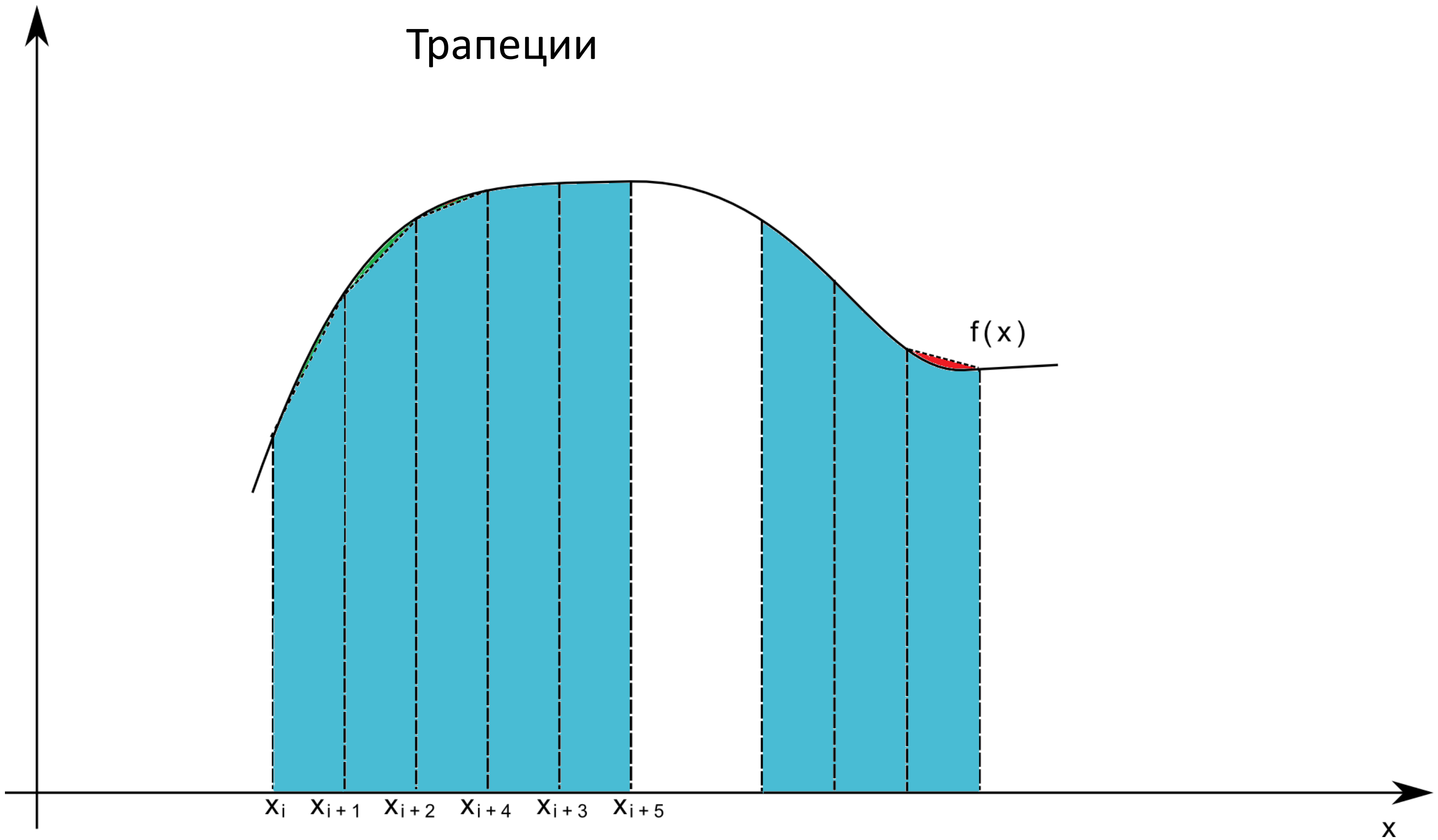
Интегрирование функций



Интегрирование функций



Чуть сложнее - трапеции



В виде формул

Прямоугольники:

- Левые $S = \sum_i^n f(x_i) \cdot h$
- Правые $S = \sum_i^n f(x_{i+1}) \cdot h$
- Трапеции $S = \sum_i^n (f(x_{i+1}) + f(x_i)) \cdot \frac{h}{2}$
- Симпсона

$$S = \sum_{i=1,2}^{n-1} [f(x_{i-1}) + 4f(x_i) + f(x_{i+1})] \cdot \frac{h}{3}$$

Сравнение

```
#include <iostream>
#include <fstream>  //  Ввод-вывод в файл
#include <vector>
#define _USE_PREDEFINED_CONSTANT
#include <cmath>
#include <stdlib.h> //  Для atof


using namespace std;

int MakeData();
int ReadData(std::vector<double>& x, std::vector<double>& y);
double IntegrRectLeft(std::vector<double>& x, std::vector<double>& y);
double IntegrRectRight(std::vector<double>& x, std::vector<double>& y);
double IntegrTrap(std::vector<double>& x, std::vector<double>& y);

int main()
{
    std::vector<double> x, y;
    MakeData();
    ReadData(x, y);
    cout << "Left rectangles:\t" << IntegrRectLeft(x,y) << endl;
    cout << "Right rectangles:\t" << IntegrRectRight(x,y) << endl;
    cout << "Trapezoid:\t" << IntegrTrap(x,y) << endl;
    return 0;
}
```

Подготовка данных

```
int MakeData()
{
    fstream inputFile;
    inputFile.open("Testdata.txt", (ios_base::in | ios_base::out) | ios_base::trunc);
    for (float i=0; i<=M_PI; i+=0.01)
    {
        cout << "x=" << i << " sin(x)=" << sin(i) << endl;
        inputFile << i << "\t" << sin(i) << endl;
    }
    inputFile.close();

}

int ReadData(std::vector<double>& x, std::vector<double>& y)
{
    fstream inputFile;
    inputFile.open("Testdata.txt");


    char buffer [1024];
    double xVal, yVal;
    int intValue;
    while (inputFile.good())
    {
        inputFile >> buffer;
        cout << buffer << "\t";
        xVal = atof(buffer);
        inputFile >> buffer;
        cout << buffer << endl;
        yVal = atof(buffer);
        x.push_back(xVal);
        y.push_back(yVal);
    }
    cout << endl;
}
```

Реализация методов

```
double IntegrRectLeft(std::vector<double>& x, std::vector<double>& y)
{
    double summ=0;
    for (int i=0; i<x.size()-1; i++)
    {
        double delta = x[i+1]-x[i];
        double value = y[i];
        summ += delta*value;
//          cout << delta << "\t" << value << "\t" << summ << endl;
    }
    return summ;
}

double IntegrRectRight(std::vector<double>& x, std::vector<double>& y)
{
    double summ=0;
    for (int i=0; i<x.size()-1; i++)
    {
        double delta = x[i+1]-x[i];
        double value = y[i+1];
        summ += delta*value;
    }
    return summ;
}

double IntegrTrap(std::vector<double>& x, std::vector<double>& y)
{
    double summ=0;
    for (int i=0; i<x.size()-1; i++)
    {
        double delta = x[i+1]-x[i];
        double value = (y[i+1]+y[i])/2;
        summ += delta*value;
    }
    return summ;
}
```

И чуть-чуть про интегрирование

```
2.98    0.160893
2.99    0.151015
3       0.141122
3.01    0.131216
3.02    0.121296
3.03    0.111364
3.04    0.10142
3.05    0.0914671
3.06    0.0815046
3.07    0.071534
3.08    0.0615562
3.09    0.0515722
3.1     0.0415831
3.11    0.0315899
3.12    0.0215935
3.13    0.0115949
3.14    0.00159517
0.00159517      0.00159517

Left rectangles:        1.99497
Right rectangles:       1.99499
Trapezoid:      1.99498
Press <RETURN> to close this window...
```

Метод Симпсона

```
int MakeData()
{
    fstream inputFile;
    inputFile.open("Testdata.txt", (ios_base::in | ios_base::out) | ios_base::trunc);
    for (float i=0; i<=1.01; i+=0.1)
    {
//          float value = sin(i);
        float value = i*i*i*i;
        cout << "x=" << i << " value=" << value << endl;
        inputFile << i << "\t" << value << endl;
    }
    cin.get();
    inputFile.close();

}

double IntegrSimp(std::vector<double>& x, std::vector<double>& y)
{
    double summ=0;
    for (int i=0; i<x.size()-2; i+=2)
    {
        double delta = (x[i+2]-x[i])/2;
        double value = (y[i+2]+4*y[i+1]+y[i])/3;
        summ += delta*value;
    }
    if (x.size()%2==0)
        summ+= (x[x.size()-1]-x[x.size()-2])*(y[y.size()-1]+y[y.size()-2])/2;
    return summ;
}
```

Метод Симпсона

```
x=0 value=0
x=0.1 value=0.0001
x=0.2 value=0.0016
x=0.3 value=0.0081
x=0.4 value=0.0256
x=0.5 value=0.0625
x=0.6 value=0.1296
x=0.7 value=0.2401
x=0.8 value=0.4096
x=0.9 value=0.6561
x=1 value=1

Left rectangles:        0.15333
Right rectangles:       0.25333
Trapezoid:      0.20333
Parabolic:      0.200013
Press <RETURN> to close this window...
```

Сделайте самостоятельно

- Проинтегрируйте с использованием двух любых методов из рассмотренных функцию $y = \ cos(x)$ на отрезке $[0, \frac{\pi}{2}]$
- Сравните точности. У какого метода она выше?
- Проведите интегрирование на отрезке $[0, \pi]$ Почему получился такой результат?

Интегрирование ОДУ

Обыкновенное дифференциальное уравнение с начальными условиями

$$\frac{du(x)}{dx} = f(x,u), \qquad y(x_0) = u_0$$

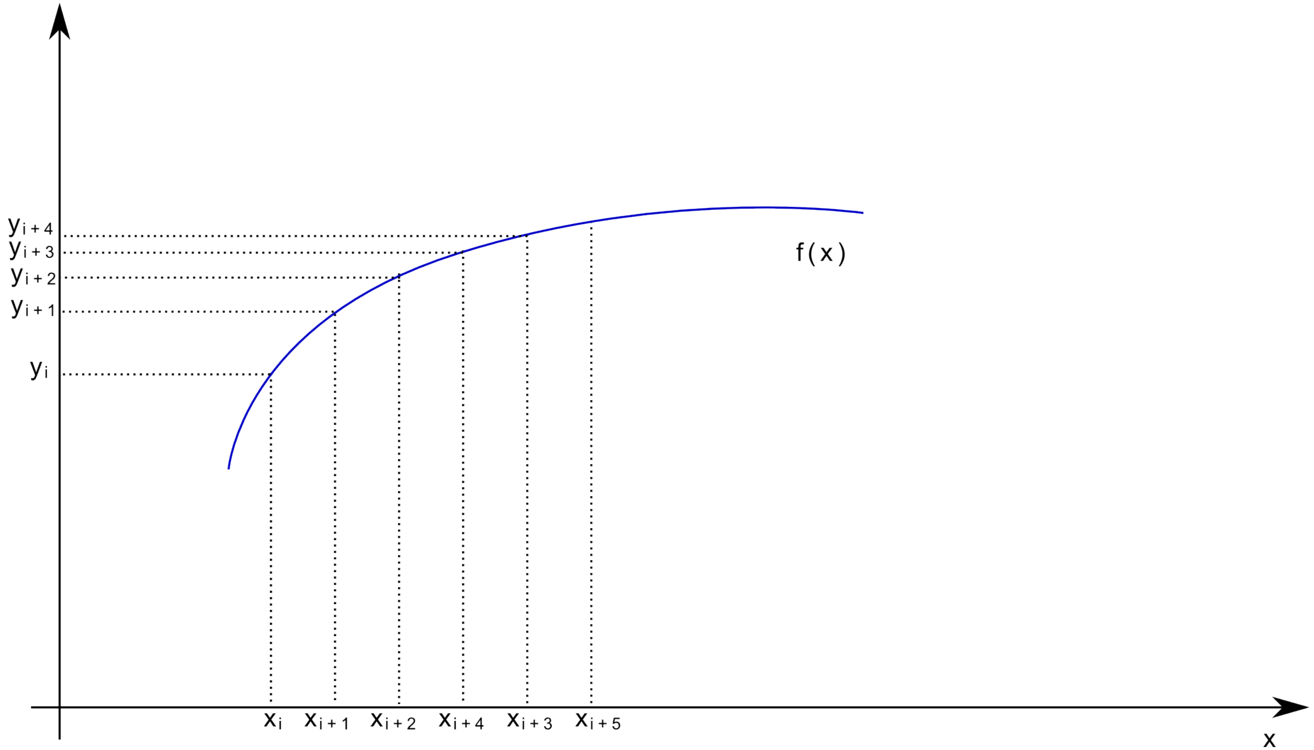
Попробуем его дискретизовать

$$\frac{du(x)}{dx} = \lim_{\Delta x \to 0} \frac{\Delta u}{\Delta x}$$

$$x_i = [x_0, x_1, x_2, \dots, x_n]$$

$$y_i = [u(x_{0)}, u(x_{)1}, u(x_{2)}, \dots, u(x_{n)}]$$

Интегрирование ОДУ



Интегрирование ОДУ

После дискретизации

$$\frac{\Delta y}{\Delta x} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Тогда, если шаг постоянен

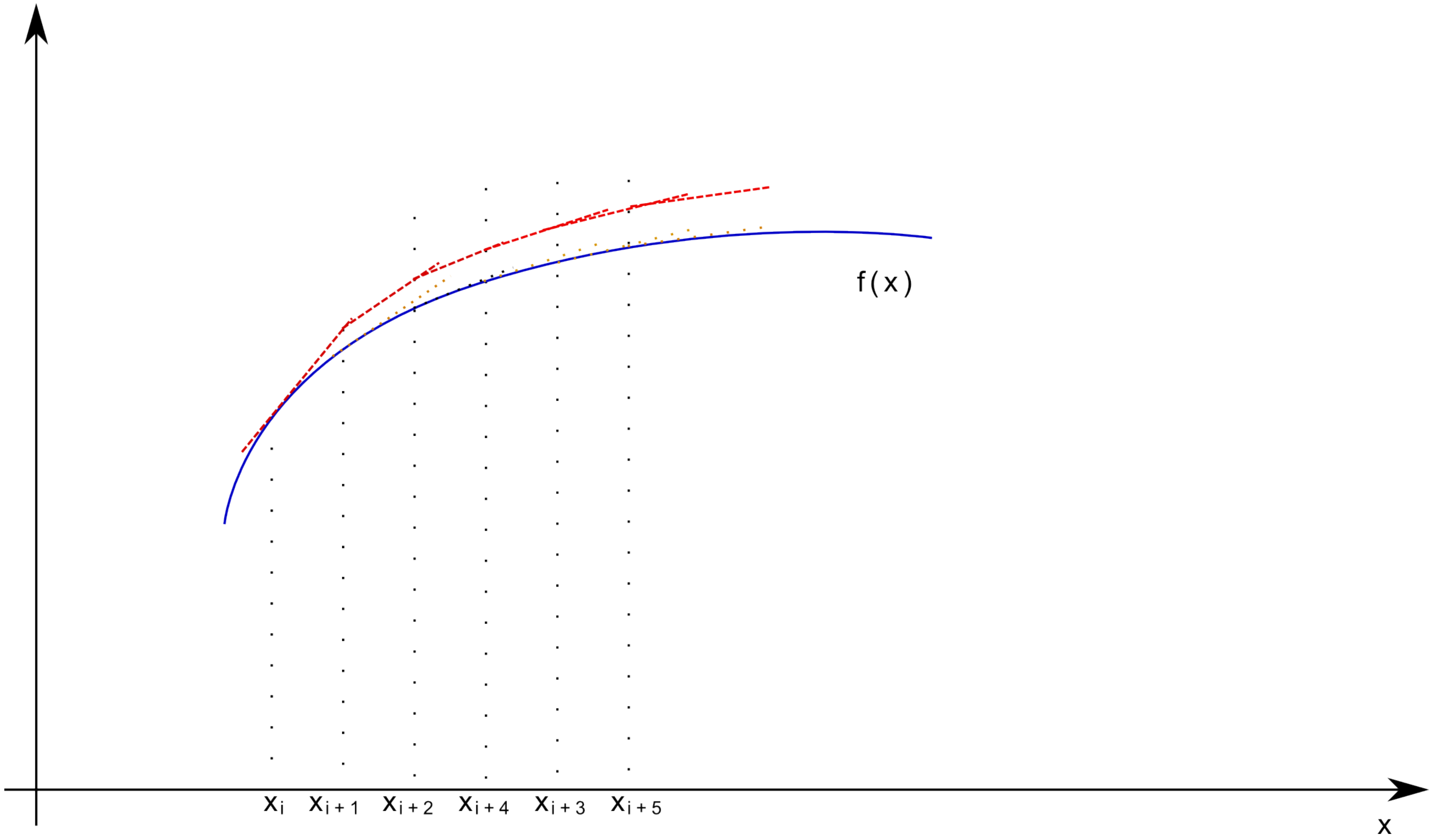
$$x_i - x_{i-1} = h_i = h$$

наше исходное уравнение можно записать

$$\frac{y_{i+1} - y_i}{h} = f(x_i, y_i), \qquad y_0 = u_0$$

Явная схема Эйлера

$$y_{i+1} = y_i + h \cdot f(x_i, y_i), \qquad y_0 = u_0$$



И немного про ОДУ

$$y_1 = y_0 + h \cdot f(x_0, y_0)$$

$$y_2 = y_1 + h \cdot f(x_1, y_1)$$

$$\dots$$

Аналогично можно получить неявную

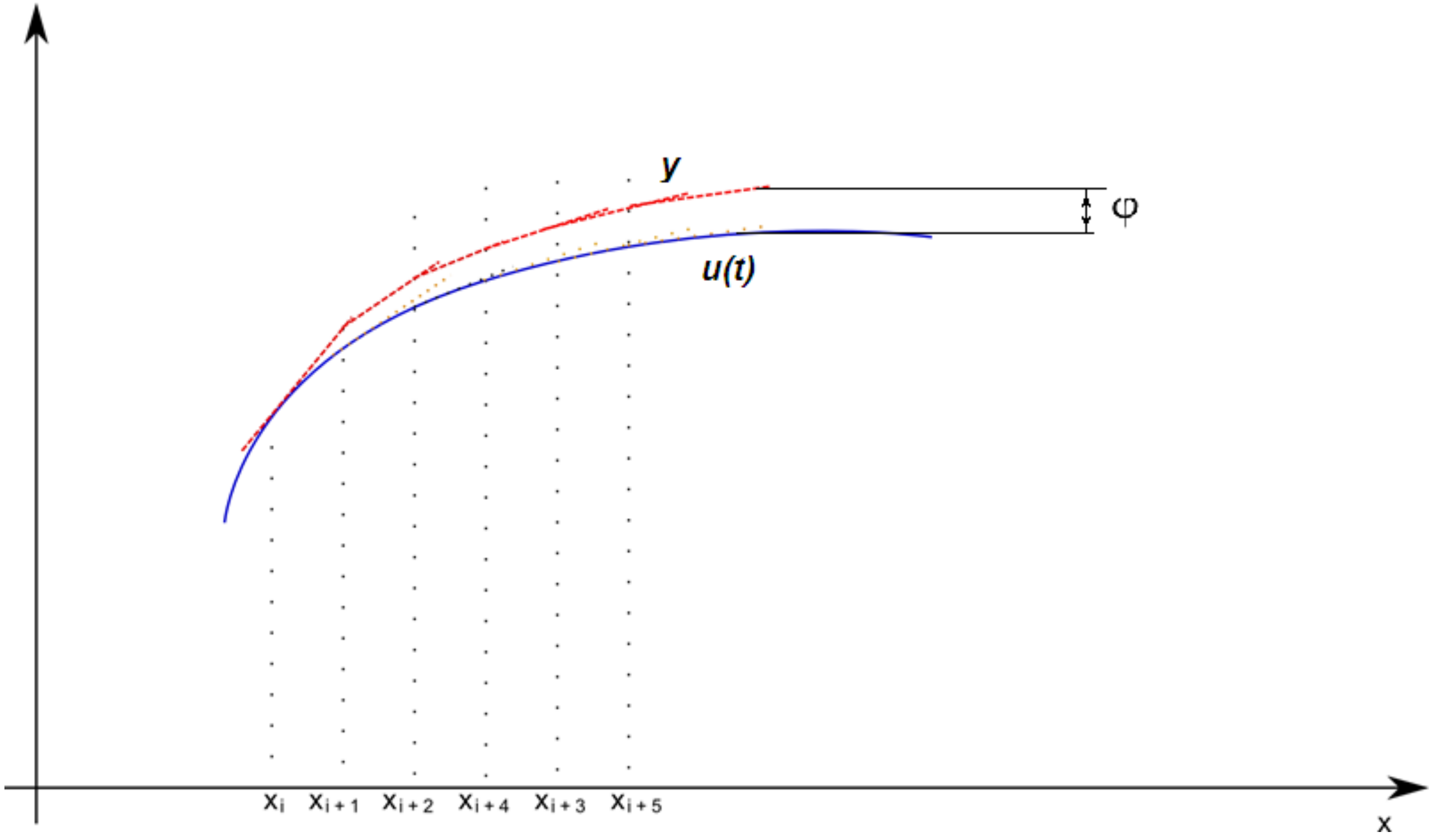
$$y_{i+1} = y_i + h \cdot f(x_{i+1}, y_{i+1}), \qquad y_0 = u_0$$

или «полуявную»

$$y_{i+1} = y_i + \frac{1}{2} h \cdot [f(x_i, y_i) + f(x_{i+1}, y_{i+1})], \qquad y_0 = u_0$$

Насколько точное решение мы получили?

$\varphi_i = u(t_i) - y_i$ – погрешность



Погрешность

Насколько точно мы получаем решение?

Для явной схемы:

$$\frac{y_{i+1} - y_i}{h} = \mathrm{y}_\mathrm{i}' + \frac{y_i''}{2h^2} + \cdots$$

Поэтому

$$\max_{i \in N} |u(x_i) - y_i| \le \max_{i \in N} |y_i''| \cdot \frac{\mathrm{h}}{2} = \mathrm{Mh}$$

Погрешность

Аналогично для полуявной схемы:

$$\frac{y_{i+1} - y_{i-1}}{2h} = \mathrm{y_i'} + \frac{y_i'''}{6h^3} + \cdots$$

Поэтому

$$\max_{i \in N} |u(x_i) - y_i| \leq \max_{i \in N} |y_i'''| \cdot \frac{\mathrm{h}^2}{6} = \mathrm{Mh}^2$$

Методы Рунге-Кутты

$$\frac{du}{dx} = f(x,u)$$

Общая идея – давайте попробуем посчитать определённый интеграл:

$$u(x_{i+1}) = u(x_i) + \int_{x_i}^{x_{i+1}} f(x,u)dx$$

Определённая сложность в том, что он обычно не вычисляем

Методы Рунге-Кутты

И снова дискретизируем:

$$x_i = [x_0, x_1, \dots, x_n]$$
$$y_i = [u(x_0), u(x_1), \dots, u(x_n)]$$

Интеграл заменим квадратурной формулой:

$$y_{i+1} = y_i + h \sum_{k=1}^{m} c_k f\left(x_i^{(k)}, y\left(x_i^{(k)}\right)\right)$$

Например, при m=2 получим метод Эйлера

Методы Рунге-Кутты

Схема Рунге-Кутты 4-го порядка точности

$$y_{n+1} = y_n + h \cdot k_n, \qquad k_n = \frac{1}{6}\left(k_n^{(1)} + 2k_n^{(2)} + 2k_n^{(3)} + k_n^{(4)}\right)$$

$$k_n^{(1)} = f(x_n, y_n), \qquad k_n^{(2)} = f(x_n + \frac{h}{2}, y_n + \frac{h}{2} k_n^{(1)}),$$

$$k_n^{(3)} = f(x_n + \frac{h}{2}, y_n + \frac{h}{2} k_n^{(2)}), \qquad k_n^{(4)} = f(x_n + h, y_n + hk_n^{(3)}),$$

В MATLAB – ode45() – с выбором шага

А.А. Амосов, Ю.А. Дубинский, Н.В. Копчёнова «Вычислительные методы для инженеров. Учебное пособие» - М., Высшая школа, 1994г.

Неявные методы Рунге-Кутты

Явные методы не подходят для решения «жёстких» задач

Неявные методы – лучше, в силу большей устойчивости

При этом задача решается итерационно

Пример – неявный метод Эйлера (он же метод Рунге 1 порядка):

$$\tilde{y}_{n+1} = y_n + h \cdot f(x_n, y_n)$$

$$y_{n+1} = y_n + h\frac{f(x_n, y_n) + f(x_{n+1}, \tilde{y}_{n+1})}{2}$$

Метод Рунге-Кутты 4 порядка

```cpp
#include <iostream>
#define USE_MATH_DEFINES
#include <cmath>
#include <vector>
using std::cout;
using std::endl;

double f (double x, double y)
{
    return (-sin(x));
}

double runge4(double t, double y0, double step);

int main(int argc, char *argv[])
{
    double initVal = 1;
    double step = 0.1;
    std::vector<double> time, y;
    time.push_back(0);
    y.push_back(initVal);
    for(double t = step; t <M_PI; t+=step)
    {
        double dy = runge4(time.back(), y.back(), step);
        time.push_back(t);
        y.push_back(dy);
    }
    for (int i=0; i<time.size(); i++)
    {
        cout << "Cos(t)=" << cos(time[i]) << " y(t)=" << y[i] << endl;
    }
    return 0;
}
```

Метод Рунге-Кутты 4 порядка

```
double runge4 (double t, double y0, double step)
{
    double k1, k2, k3, k4;
    k1 = f(t, y0);
    k2 = f(t+step/2, y0+step*k1/2);
    k3 = f(t+step/2, y0+step*k2/2);
    k4 = f(t+step, y0+step*k3);
    double y = y0 + step*(k1+2*k2+2*k3+k4)/6;
    return y;
}
```

```
Cos(x)=1 val=1
Cos(x)=0.995004 val=0.985062
Cos(x)=0.980067 val=0.960332
Cos(x)=0.955336 val=0.926057
Cos(x)=0.921061 val=0.882578
Cos(x)=0.877583 val=0.830331
Cos(x)=0.825336 val=0.769838
Cos(x)=0.764842 val=0.701703
Cos(x)=0.696707 val=0.626606
Cos(x)=0.62161 val=0.545298
Cos(x)=0.540302 val=0.458592
Cos(x)=0.453596 val=0.367354
Cos(x)=0.362358 val=0.272495
Cos(x)=0.267499 val=0.174963
Cos(x)=0.169967 val=0.075733
Cos(x)=0.0707372 val=-0.0242037
Cos(x)=-0.0291995 val=-0.123849
Cos(x)=-0.128844 val=-0.222206
Cos(x)=-0.227202 val=-0.318294
Cos(x)=-0.32329 val=-0.411151
Cos(x)=-0.416147 val=-0.49985
Cos(x)=-0.504846 val=-0.583505
Cos(x)=-0.588501 val=-0.66128
Cos(x)=-0.666276 val=-0.732398
Cos(x)=-0.737394 val=-0.796148
Cos(x)=-0.801144 val=-0.851893
Cos(x)=-0.856889 val=-0.899076
Cos(x)=-0.904072 val=-0.937227
Cos(x)=-0.942222 val=-0.965962
Cos(x)=-0.970958 val=-0.984997
Cos(x)=-0.989992 val=-0.994139
Cos(x)=-0.999135 val=-0.993299
Press <RETURN> to close this window...
```

Самостоятельно:

Давайте попробуем проинтегрировать

$$y' = y, y(0) = 1$$

Что должно получиться?

Как решить более сложное уравнение?

Мы с вами выяснили, как решить уравнение

$$\frac{du}{dt} = f(t,u), u(t_0) = u_0$$

А как нам быть, если уравнение несколько сложнее, например:

$$(m + \lambda_{11}) \cdot \ddot{x} + C_1\dot{x}|\dot{x}| + C_2\dot{x} - F = 0, \qquad x(t_0) = u_0, v(t_0) = 0$$

Метод Рунге-Кутты – для уравнений первого порядка

Система уравнение

Перейдём к системе уравнений:

$$\begin{cases} \dot{x} = v, \\ (m + \lambda_{11}) \cdot \dot{v} + C_1 v|v| + C_2 v - F = 0 \end{cases}$$

С начальными условиями $x(t_0) = u_0, v(t_0) = 0$

К чему это приведёт с точки зрения кода?

Функции правых частей

```
#include <iostream>

#include <vector>

using namespace std;

double B1 = 1,

B2 = 1,

F = 10;

double f1(double x, double v, double t)

{

        return v;

}
double f2(double x, double v, double t)

{

        return (-x);

        // return (-B1*v*v - B2*v + F);

}
```

Класс-решатель

```
class Runge4Solver
{
public:
        Runge4Solver(double x, double v, double t)
        {
                m_x.push_back(x);
                m_v.push_back(v);
                m_time.push_back(t);
        }
        double x() const {return m_x.back();}
        double v() const {return m_v.back();}
        double t() const {return m_time.back();}
        void CalcStep()
        {
                …
        }
private:
        std::vector<double> m_time, m_x, m_v;
        double k_x[4], k_v[4];
        double m_step = 0.01;
};
```

CalcStep

```
void CalcStep()
{
        double x_prev = m_x.back(),
        v_prev = m_v.back(),
        t_prev = m_time.back();
        k_x[0] = f1(x_prev, v_prev, t_prev);
        k_v[0] = f2(x_prev, v_prev, t_prev);
        k_x[1] = f1(x_prev+k_x[0]*m_step/2, v_prev+k_v[0]*m_step/2, t_prev+m_step/2);
        k_v[1] = f2(x_prev+k_x[0]*m_step/2, v_prev+k_v[0]*m_step/2, t_prev+m_step/2);
        k_x[2] = f1(x_prev+k_x[1]*m_step/2, v_prev+k_v[1]*m_step/2, t_prev+m_step/2);
        k_v[2] = f2(x_prev+k_x[1]*m_step/2, v_prev+k_v[1]*m_step/2, t_prev+m_step/2);
        k_x[3] = f1(x_prev+k_x[2]*m_step, v_prev+k_v[2]*m_step, t_prev+m_step);
        k_v[3] = f2(x_prev+k_x[2]*m_step, v_prev+k_v[2]*m_step, t_prev+m_step);
        m_x.push_back(x_prev+m_step*(k_x[0]+2*k_x[1]+2*k_x[2]+k_x[3])/6.0);
        m_v.push_back(v_prev+m_step*(k_v[0]+2*k_v[1]+2*k_v[2]+k_v[3])/6.0);
        m_time.push_back(t_prev+m_step);
}
```

main()

```
int main()
{
        Runge4Solver system(0,1,0);
        while (system.t() < 3.15)
        {
                system.CalcStep();
                cout << "X=" << system.x() << "\tV=" << \
                system.v() << "\tT=" << system.t() << endl;
        }
        return 0;
}
```

Можно ли сделать более универсальное решение?

- У нас жёстко задан размер системы уравнений
- Снаружи класса - какие-то непонятные функции *f1* и *f2*
- Как быть, если мы захотим сменить систему уравнений?
- *Вспомним про возможность наследования!*

Базовый класс с виртуальным методом

```
class Runge4Solver
{
public:
        Runge4Solver(){}
        void InitValues (std::vector<double>& initVals, double initTime)
        {
        m_values = initVals;
        m_time = initTime;
        }
        void CalcStep()
        {
                …
        }
        double t() const {return m_time;}
        std::vector<double> vals() const {return m_values;}

protected:
        virtual std::vector<double> RecalcSystem(double time) =0;
        std::vector<double> m_values;
        double m_time;
        double m_step = 0.01;
};
```

Собственно, интегрирование

Здесь мы несколько раз вызываем виртуальную функцию RecalcSystem()

```
void CalcStep()
{
	std::vector<double> tmp, partialVals(m_values.size()), previousVals = 
m_values;
	tmp = RecalcSystem(m_time);
	for (int i = 0; i<tmp.size(); i++)
	{
		partialVals[i] = tmp[i];
		m_values[i] = previousVals[i]+tmp[i]*m_step*0.5;
	}
	tmp = RecalcSystem(m_time+m_step*0.5);
	for (int i = 0; i<tmp.size(); i++)
	{
		partialVals[i] += 2*tmp[i];
		m_values[i] = previousVals[i]+tmp[i]*m_step*0.5;
	}
	tmp = RecalcSystem(m_time+m_step*0.5);
	for (int i = 0; i<tmp.size(); i++)
	{
		partialVals[i] += 2*tmp[i];
		m_values[i] = previousVals[i]+tmp[i]*m_step;
	}
	tmp = RecalcSystem(m_time+m_step);
	for (int i = 0; i<tmp.size(); i++)
	{
		partialVals[i] += tmp[i];
		partialVals[i] *= m_step/6.0;
		m_values[i]= previousVals[i]+partialVals[i];
	}
	m_time+=m_step;

}
```

А вот и класс наследник

В нём реализована та самая функция

```
class TestRunge4 : public Runge4Solver
{
        double B1=1,
        B2=1,
        F=10;
        virtual std::vector<double> RecalcSystem(double time)
        {
        std::vector<double> ret(m_values.size());
        ret[0] = m_values[1];
        ret[1] = -B1*pow(m_values[1],2) - B2*m_values[1] + F;
        return ret;
        }
};
```

И ещё один

Теперь я могу создать кучу классов – под каждую систему арвнений

```
class MyRunge4 : public Runge4Solver
{
        virtual std::vector<double> RecalcSystem(double time)
        {
                std::vector<double> ret(m_values.size());
        ret[0] = m_values[1];
                ret[1] = -m_values[0];
                return ret;
        }
};
```

main()

```
int main()
{
        MyRunge4 system;
        std::vector<double> init;
        init.push_back(0);
        init.push_back(1);
        system.InitValues(init, 0);
        while (system.t() < 3.15)
        {
                system.CalcStep();
                std::vector<double>vals = system.vals();
                cout << "X=" << vals[0] << "\tV=" << \

                vals[1] << "\tT=" << system.t() << endl;
        }
        return 0;
}
```

Третье задание

Будем решать уравнение $\frac{dy}{dt} = at - by$

Начальное условие $y(0) = d$

Решаем на интервале $[0,1]$ с шагом 0.01

Аналитическое решение $\mathrm{u}(t) = \frac{a}{b}\left(x - \frac{1}{b}\right) + C \cdot e^{-bx}$

Очевидно $-\frac{a}{b^2} + C = d$

Нужно решить и вывести значения

и максимальную величину рассогласования $|u(t) - y(t)|$

	a	b	d
1	1	0.2	1
2	-2	0.2	1
3	0.3	0.2	1
4	-0.7	0.7	1
5	1.2	0.7	0
6	-0.8	0.7	0
7	2.5	2.1	0
8	-2	2.1	0
9	0.3	2.1	0.5
10	-0.7	1.3	0.5
11	1.2	1.3	0.5
12	-0.8	1.3	0.5
13	2.5	1.3	0.5